Comparison of mine to Saldamlis extension

November 1, 2013

According to 4th chapter in theses of Levon Saldamli.

4.2.1 Field Variables – used

4.2.2 Field Constructor – used, slightly modified – `field(expr in domainName.regionName);` in `expr` appears only coordinate variables defined in `domainName`

4.2.3 Field Type in Expressions

- 4.2.3.1 **Binary operators** – used

- 4.2.3.2 **Special operators** – for time derivative of field should be used pder operator instead of der

4.2.4 Accessing Field Values – Accesing field values in function-like style should not be allowed, if possible, for two reasons:

- it is not allowed in current Modelica for regular variables (unknown functions of time)
- If more then one coordinate system are defined in a domain, it is not clear which coordinates are used in the function-like expression

Regions consisting of one point and the `in` operator will be used instead. e.g.

```
model heatPID
  record Room extends DomainBlock3D;
    Region0D sensorPosition(shape = shapeFunc, range = {{1, 1},
{0.5, 0.5}, {0.5, 0.5}});
  end Room
  Room room(...)
  field Real T(domain = room);
  Real Ts;
  ...
equation
  Ts = T in room.sensorPosition;
  ...
end heatPid;
```

4.3.1. Domains Geometry Deffinition

Main difference: Saldamli describes boundary by curves in 2D resp. surfaces in 3D and thus specifies the domain shape.

This approach doesn't work very well in 3D. If boundary is made of several surfaces in 3D, parameters (arguments) of shape-functions of these surfaces would heve to be bounded not just in cartesian product of intervals but in some more complex sets to compose continuous boundary. And there is no way to write this in Levons extension.

I prefare the way used in Peters book – write a function (or equation) that maps an cartesian product of intervals onto the interior and other regions of the domain. I don't think this approach can describe more complicated geometries in 3D, but is closer to the way how such a region (subsets of R^n) is usualy described in mathematics. I think it will be aslo easier to generate the computational grid if we have a function or equation that maps cartesian product on the domain.

There are two ways for the seccond approach:

- using shape-function (used in the book)
- using equations holding coordinate transformations and define always some general coordinates that are bounded in cartesian product of intervals

For illustration see domain_comparison.lyx/pdf file.

This way of domain deffinition should be supplemented with Constructive Solid Geometry – it is building more complex domains using union, intersection and difference of previously defined domains. This is not designed already. It should be also possible define domain in external file from some CAD app.

4.3.1.1 **Domain Type** similar (built-in) - In both cases all domains extends general built-in `Domain` type. Saldamlis built-in `Domain` has different members. We have only `replaceable Region interior;`

3.2.1.2 **Boundary type** (built-in) – we don't needed it. Here is `Region0D, Region1D, Region2D` and `Region3D` built-in type to describe boundaries, interior and other regions in the domain instead.

4.3.1.3 **Coordinate Systems** Saldamlis domain always has cartesian coordinates defined. I would not insist on this.

I'm not shure if we shell have a spetial data type `Coordinate`, or it should be of type `Real` and compiler itself would infere that a variable is a coordinate that must be treated in a special way. Other option is to consider the variable type as somethink different from a data type because coordinate variable indeed doesn't hold any data and has a symbolic meaning (somethink like `time`). Than coordinates could be perhaps defined using keyword with lower case:

`coordinate x;`

4.3.2 Differential Operators

4.3.2.1 **Partial derivative** Saldamli uses `der(u,x,y..)`, here `pder(u,x,y..)`. Time derivatives should not be written as `(p)der(u)` but `also pder(u,time)`.

4.3.2.2 **Normal derivative** (or rather normal vector) - similar, unfinished design. It perhaps should be member of the region, e.g.:

`pder(u,omega.boundary.n) = 0 in omega.boundary;`

or an operator applied on a region, somethink like:

`pder(u,normal(omega.boundary)) = 0;`

Writing boundary conditions using normal vector but outside differentiation should be also allowed e.g.:

```
field Real[3] flux;
flux*region.n = 0 in omega.boundary; //*scalar product
```

4.3.2.3 **Vector notation** – coordinate-free differential operators as `gradient`, `divergence`, `rotation` ... I didn't work up how to define this. Could be accepted.

4.3.3 Domain Specifier in Equations

keyword `in` accepted. Should be used not only for boundary conditions but also for PDE to specifie region and to acces field values in particular points (example above).

4.3.4 Field Reduction – integral operator. I didn't consider yet, coul'd be perhaps undertaken.

Other notes:

Equations may be written within or outside domain class. Outside domain coordinates must be accesd of course `domainName.coordName`. To make a shortcut we introduced `region` and `domain` (or `dom`) keywords as a alias for domain and region specified after `in` op.

To acces region we can write

```
pder(u,region.n) = 0 in omega.boundary;
```

similarly for domain

```
pder(u,dom.x) in omega.interior;
```

It shoud be possible to write equations that relates field defined in different domains. This is not designed yet.